CS10720 Problems and Solutions

Thomas Jansen

Today: Towards Computability

April 11th

Plans for Today

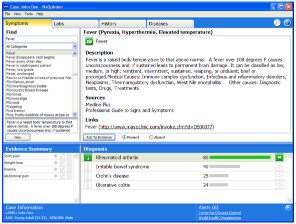
What Computers Are Doing

chess computer

robot vacuum cleaner

autonomous car

high frequency trading



medical diagnosis

drone control

What Computers Can't Do

Careful — with predictions about what computer's can or cannot do

Facts
- AI researchers have routinely overestimated what computers will be able to do in a few years
  e. g., Herbert Simon predicted in 1957 that by 1967
  - computers become world chess champion (actually 1997)
  - computers discover important new mathematical theorems (played a role in 1976, never alone)
  - computers become the standard way of describing theories in phsychology (psychology moved away from this)
- AI critics have routinely used vague and non-falsifiable statements about what computers cannot do
  e. g., doubting 'true consciousness' without precisely defining what this is and how it can be determined

Consider — computers making decisions with ethical component and ask can computers be responsible in an ethical sense?

Take note — increasing number of people see AI as existential thread for mankind
e. g., Elon Musk, Stephen Hawking

Topic 'What Computers Can't Do' in CS107

Goal for us here
restrict ourselves to computational problems
determine if those can be solved by computers

What do we mean by computational problem?

Computational problem has some finite input
and is solved by some finite, correct output

Examples

- Sorting Input items with keys
  Output same items sorted in ascending order according to keys
- Addition Input two numbers in two's complement
  Output one number in two's complement that is the sum of the two input numbers
- Chess Input chess board w. pieces in valid pos., colour
  Output best move for the active side

Computational Problems

Remember we can encode everything in binary
using only the 'alphabet' $\{0, 1\}$

Fact sometimes more convenient larger alphabets
e. g., $\{a, b, c, \ldots, z, 0, 1, 2, \ldots, 9, \_\}$
but important always using finite alphabet with at least 2 letters

Computational problems have some input
(over some finite input alphabet)
and are solved by some output
(over some finite output alphabet)
input alphabet and output alphabet
could be different or equal

Computational Problems

Definition (Computational Problem)

A computational problem is defined by a set of finite inputs over a finite input alphabet and for each input a set of correct finite outputs over a finite output alphabet.

Example
- input directed graph with edge weights, nodes A and B
- output shortest path from A to B

Definition (Optimisation Problem)

A optimisation problem is a computational problem where the output is the value of an optimal solution.

Example
- input directed graph with edge weights, nodes A and B
- output length of shortest path from A to B

Definition (Decision Problem)

A decision problem is a computational problem where the output is ‘yes’ or ‘no’ (alternatively, ‘1’ or ‘0’ if we prefer binary encodings).

Example
- input directed graph with edge weights, nodes A and B, value k
- output yes if there is a path from A to B of length $\leq k$, no otherwise

Solving a Problem with Help of the Optimisation Problem

Remember shortest path problem

Assume have efficient algorithm for solving optimisation variant

```
Algorithm   to solve the computational problem
Compute opt. length l using algo. for opt. problem.
For each edge e {
  Set w := weight(e).
  Set weight(e) := l + 1
  Compute opt. length m using algo. for opt. problem.
  If m > l then set weight(e) := w.
}
Output all edges with weight ≤ l.
```

Observation is simple
and not much slower than algo. for opt. problem

Solving a Opt. Problem with Help of the Decision Problem

Remember shortest path problem

Assume have efficient algorithm for solving decision variant

Algorithm to solve the computational problem

```
Compute S := sum of all edge weights.
If solution with length ≤ S exists then {
  Set l := 0 and u := S.
  while l < u {
    Set m := ⌈l + (u − l)/2⌉.
    If solution with length ≤ m exists
    then set l := m else set u := m − 1.
  }
  Output l
}
else Output 'no path from A to B'
```

Observation is simple
and not much slower than algo. for decision problem

Reductions

Fact Solving some problem P
using an algorithm for some other problem Q
is called 'reducing P to Q'
and implies in some sense P is not much harder than Q

Special case solving decision problem P
using an algorithm for some other decision problem Q
is called 'reducing P to Q' and written as $P \leq Q$
and implies in some sense P is not much harder than Q

Observation reductions can help to find
more problems that computers cannot solve
If we know that P cannot be solved by computer
and if we know that $P \leq Q$
then Q cannot be solved by computer
because we can solve P with the help of Q
and the '$\leq$'-algorithm

Towards Our Goal

Remember we want to find out
what kind of computational problem
computers cannot solve

What do we mean by 'cannot solve'?

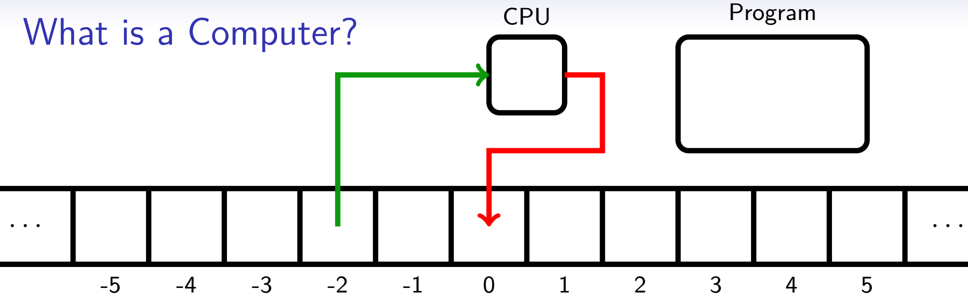
What we do NOT mean is

- cannot be solved because of lack of resources (memory, time, ...)
- cannot be solved because of lack of a clever idea
- cannot be solved because we do not quite understand problem

What we DO mean is
cannot be solved for principle reasons
not today, not tomorrow—never

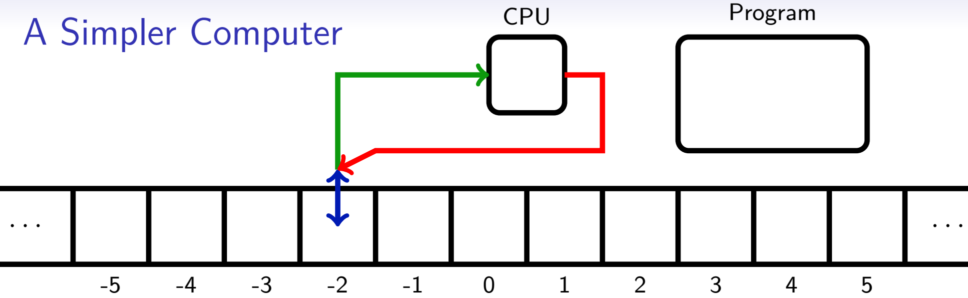
Observation need precise idea of what a computer is
to determine what computers cannot do

What is a Computer?



- huge number of memory cells (to avoid problems with lack of memory, let's say infinitely many)
- CPU that contains a little memory
  (we formalise this as a 'state' the CPU is in)
- program the CPU follows (by executing it)
- CPU operates in steps (one command of program in one step)
- in each step, CPU can read from one cell of memory
- in each step, CPU can write to one cell of memory
- in each step, CPU can change its state

A Simpler Computer



- infinite number of memory cells
- CPU that is in a state
- program the CPU follows (by executing it)
- CPU operates in steps (one command of program in one step)
- in each step, CPU can read from current cell of memory
- in each step, CPU can write to the same cell of memory
- in each step, CPU can change its state
- in each step, current cell can change to neighbouring cell

Observation can solve the same problems (is just slower)

Turing Machine

Definition (Turing Machine)

A Turing machine has a finite set of possible states Q, an initial state $q_0 \in Q$, a finite memory alphabet Γ that contains the blank character _, a finite input alphabet Σ that does not contain _, an infinite memory that is linearly organised, and a current position in the memory. Initially the input is in the memory, the current position is the first position of the input and all unused memory cells contain _. Its functioning is defined by a program $P\colon Q \times \Gamma \to Q \times \Gamma \times \{L, R, *\}$. It operates in steps, in each step it is in some state $q \in Q$ and reads the contents of the current cell $a \in \Gamma$. If $P(q, a) = (r, b, d)$ then it replaces a by b, changes state from q to r and changes the current cell to its neighbour if $d = L$, to its right neighbour if $d = R$, leaving it unchanged if $d = *$.

Observe Turing machine is our simpler computer
can solve exactly all problems that any computer can solve

Ends of Computation

How do we know that a Turing machine has ended its computation?

Different options

- define some state $q_f \in Q$ to be stop state
- machine reads $a \in \Gamma$ and is in $q \in Q$ and $P(q, a) = (q, a, *)$.

Where do we find the output?

Different options

- written in memory
- for decision problems, define two special stop states q_{yes} and q_{no} and let state define the result of the computation

Summary & Take Home Message

Things to remember

- limits to what computers can do: non-computational problems
- computational problems, optimisation problems, decision problems
- reductions
- formalising 'computer': the Turing machine

Take Home Message

- Computers solve computational problems (and their use should be restricted to such problems).
- Reductions help to compare the difficulty of computational problems.
- The Turing machine is a formal model of computations that helps us explore what computer can and cannot compute.

Lecture feedback `http://onlineted.com`